Billboard shader pack documentation

Contents

1 Introduction

1.1 What are billboards anyways?

This section is shamelessly copied from [WIKI]

In computer graphics, billboards are textured rectangles that are transformed such that they always appear parallel to the view plane. Thus, they are similar to billboards along highways in that they are rotated for best visibility. However, they are different from highway billboards since they are dynamically rotated to always offer best visibility.

The main use of billboards is to replace complex three-dimensional models (e.g. grass, bushes, or even trees) by two-dimensional images. In fact, Unity also uses billboards to render grass. Moreover, billboards are often used to render two-dimensional sprites. In both applications, it is crucial that the billboard is always aligned parallel to the view plane in order to keep up the illusion of a three-dimensional shape although only a two-dimensional image is rendered.

1.2 Why did you create this package?

While unity does use the billboard thechnique for rendering grass, distant trees and particles, it offers no direct way for us users to create billboards directly. When we wanted to create billboards we had to make use of workarounds. One way to fake billboards I found was rotating a plane towards the camera every frame [BILL], which seems like lots of unnecessary work and ultimately fail if there are more two or more cameras.

The best solution seemd to be a billboard shader. However neither the wiki nor the asset store had a free, ready to use billboard shader available.

Thus I decided to dig myself into shader scripting and work something out on my own. Prior to this I had no experience with programming shaders at all and I had some respect for it. But I had already taught myself scrip and there is some documentation [WIKI] to be found so I just gave it a shot.

I was pretty satisfied with the results of my work. Therefore I felt obliged to share the results with the unity community. Hopefully you find them usefull.

1.3 May I use this package for free?

Yes!

This pack is published into the public domain. You are free to use it in any commercial or non-commercial work or create drived work form it.

Although not required, it would be appreciated if you would credit me when you use this pack in your project or create new shaders based on mine [GRIMM].

If you feel that crediting is not enough, feel free to make a donation from on my homepage [GRIMM]. Again: This is totally not required! But I would feel honored if you did.

2 How to use the shaders

To use the shaders you have to create a new material. Click the shader property in the inspector to bring up the dropdown menu. Hover over *Billboard* and select your desired shader from the submenu. Detailed information on how to set up the specific shader is providet below.

Next create a cube and apply your new material. And you are already done.

You can scale the billboard by manipulating the scale the cube. The width will respond to the x-scale and the height will respond to the y-scale. Make sure to leave the z-scale at 1 (see 3).

You can also use other meshes with the billboard shader. In example: You can use a sphere and the *cutoff color* shader to create crecent moon like in the example. Feel free to play with other shapes.

2.1 Simple opaque

This is the most simple billboard shader in the pack. It will simply render the texture assigned to it.

This shader supports tiling and offsetting. It does not support transparency.

2.2 Simple opaque color

Same as *Simple opaque* but allows coloring.

2.3 Simple transparent

Same as *Simple opaque* but with transparency.

2.4 Simple transparent color

Same as *Simple opaque* but with transparency and color.

2.5 Cutoff color

This shader will render a solid color based on its ramp textures alpha channel.

It is a cutoff shader thus it will only render fully opaque or fully transparent. The breakpoint is defined by the *Alpha Cutoff* parameter. You can manipulate this parameter via script by using: `renderer.material.SetFloat("‘_Cutoff"’, somefloat);`

I found it most convenient to use a black and white texture and set *alpha from grayscale* to true. Make sure to set the *wrap mode* to clamp if you use a linear white to black ramp.

This shader also supports tiling and offsetting. However the *wrap mode* must be set to repeat for this to work properly.

2.6 Cutoff textured

This shader is a bit of a combination of the previous ones.

Instead of a solid color it renders a texture which gets cut out based on its alpha channel. Like the *Cutoff color* shader this can be regulated via script with `renderer.material.SetFloat("‘_Cutoff"’, somefloat);`

2.7 Alphamap

This shader works like the *Simple transparent* shader but supports an additional alpha map.

The alpha values of both textures get multiplied (values are between 0 and 1). Frankly spoken: the transparency adds up.

This is very convenient if you want your billboard to have round edges without editing your picture. Also tiling works independently on the texture and the alpha map which can create some interresting results.

I found it most convenient to use a black and white texture and set *alpha from grayscale* to true.

2.8 Alphamap colored

Same as *alphamap* but with color property.

2.9 Progressbar

This is the most advanced shader in this package. Its intended use is (what a surprise) displaying a progress bar.

Since it is also the most specific shader, setting it up requires a bit more consideration. This shader makes clever use of the all the technique from the shaders above. Lets look at what it does in detail:

First *frame texture* gets drawn. Give it an alpha channel to get some nice round edges. Use the *frame color* to get a nice tint.

Then the *bar texture* gets drawn above it. It must be set up accordingly to the *frame texture*. Also give it an alpha channel to fits nicely into the frame.

Finally the *bar texture* gets cropped depending on the *ramp* and the *progress*. Just like the cutoff shaders do. For a classic progress bar use a white to black gradient and set *alpha from grayscale* to true and*wrap mode* to clamp.

The *progress* property can be set up via script with
`renderer.material.SetFloat("‘_Progress"’, somefloat);`

Tiling is not supported for this shader.

2.10 3D Text

This shader has been modified to work with the quads generated by the textmesh component. Due to their setup flipping both the x and the y direction was required. Since there is no controll over these quads, this shader might fail if the textmesh component gets updated by unity. Use it at your own risk.

Setting up the material works just as with the regular 3D-text material. Import your ttf and assing the automatically created font texture to your material. Note that the color property textmesh component will have no effect. You have to use the color property of the material instead.

Setting the *anchor* property to *middle center* seems to produce best results. To avoid blurriness use the standart trick of scaling the fontsize up and the gameobject down.

3 Known issues

When two billboards with the same material are close to each other they will be displayed at an arbitrary point.
What triggers effect is dependant on the position of the rendering camera. However this does not occur when both gameobjects have a different material. Even if both materials use the same shader. I have no idea what causes this.

When the scale of the cube is a multiple of the vector (1,1,1) it will ignore the scale.
The billboard takes the gameobject's x and y scale to scale themself. Just leave the z scale at 1 and you should be fine.

4 References

[WIKI] http://en.wikibooks.org/wiki/Cg_Programming/Unity/Billboards

[BILL] http://wiki.unity3d.com/index.php?title=CameraFacingBillboard

[GRIMM] Benjamin Grimm http://grimmreapergames.jimdo.com/